1989

Rapid Prototyping Languages for Expert Systems

Luqi

Naval Postgraduate School

NPS52-89-032

NAVAL POSTGRADUATE SCHOOL

Monterey, California

RAPID PROTOTYPING LANGUAGES FOR EXPERT SYSTEMS

LUQI

March 1989

Prepared for:

Naval Postgraduate School
Monterey, CA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

H. Shull
Provost

The work reported herein was supported by the National Science Foundation, the Office of Naval Research and the Naval Postgraduate School Research Council.

This report was prepared by:

[redacted]
LUQI
Assistant Professor
of Computer Science

Reviewed by:

[redacted]
ROBERT B. MCGHEE
Chairman
Department of Computer Science

Released by:

[redacted]
KNEALE T. MARSHALL
Dean of Information
and Policy Science

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-89- 032	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable) 52	7a. NAME OF MONITORING ORGANIZATION National Science Foundation & ONR Sponsored Navy Direct Funding
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code) Washington, D. C. 20550
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Postgraduate School 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER O&MN, Direct Funding NSF CCR-8710737
8c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

RAPID PROTOTYPING LANGUAGES FOR EXPERT SYSTEMS (U)

12. PERSONAL AUTHOR(S)
LUQI

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Progress	FROM Sept 88 TO Mar 89	1989 March	14

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Darpa/ISTO is seeking to develop designs for a new language for rapid prototyping. The language is seen as part of longer subsequent efforts to develop a comprehensive prototyping system that will provide additional tools realizing a high-productivity software design and prototyping environment. This report presents the concepts of a prototyping language and relations to Expert Systems.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL LUQI	22b. TELEPHONE (Include Area Code) 408-646-2735	22c. OFFICE SYMBOL 52Lq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-24

Rapid Prototyping Languages for Expert Systems

Luqi

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

1. Introduction

Work on rapid prototyping languages is aimed at reducing software development costs via prototyping. A software prototype is an executable initial version of a proposed system. Prototypes are built to assess whether a proposed system will be acceptable to its users and whether a proposed design will provide adequate functionality and performance. A prototype is constructed prior to the production version of the system for the purposes of

(1) gaining information to guide analysis and design, and

(2) supporting the generation of the production version.

To be useful, the prototype must be constructed quickly and at low cost. Thus a prototyping language must make it easy to construct, modify, and instrument prototypes, possibly at the expense of efficiency, completeness, capacity, or robustness. Prototyping is especially useful for large systems or novel application areas. Expert systems often fall in the latter category.

Rapid prototyping languages are intended to support a comprehensive set of tools for computer-aided software design and prototyping. The goals for such a language and the associated prototyping system are:

(1) Rapid construction and adaptation of software,

(2) Enabling the development of more powerful systems,

(3) Checking if specified systems are acceptable to users,

(4) Checking internal consistency of proposed designs, and

(5) Ensuring that implementations conform to specifications.

In this paper we discuss the principles of language support for rapid prototyping, based on our experience with the design of a prototyping language and the feasibility study for its implementation over the past five years. We examine some of the basic issues involved in the design of a rapid prototyping language, with attention to issues involving the prototyping of expert systems.

2. Requirements for a Comprehensive Prototyping Language

The goal of developing a general purpose prototyping language is very ambitious, and requires solutions to some open research problems for complete fulfillment. Prototyping has potential benefits for large software systems, many of which are concurrent, distributed, and exhibit hard real-time constraints. Expert systems are being implemented with concurrent and distributed configurations, and there is a growing demand for expert systems capable of meeting hard real-time constraints. In this section we discuss the general properties of a comprehensive prototyping language.

A prototyping language should have a clear and simple structure and semantics to make it easy to learn, understand, and process mechanically and rapidly. This implies uniform structure, a small number of orthogonal constructs, and general interpretations without special cases or restrictions. To support automated tools, the language should have an abstract syntax and an unambiguous and precisely defined meaning. The underlying model should have a mathematical basis to support execution, analysis, verification, and trusted transformations. In particular, the semantics of the language should support rigorous reasoning about the properties of prototypes described in the language and transformations on expressions of the language. The language should also support a user interface for communication with untrained people, with graphical summary views, English paraphrasing, and explanation facilities.

A prototyping language should be expressive. It should be easy to use the language to construct concise and clear descriptions for a wide variety of systems. This implies language support for abstractions, uniform communication, logical inference, incomplete descriptions, and automated design completion. In addition to providing traditional facilities for functional, data, and control abstraction, the language should also support abstractions for concurrency, synchronization, and timing constraints. The language should be at a specification and design level rather than at a programming level: the constructs of the language should correspond directly to decisions made by the designer, rather than to operations performed by the processor. This will make prototype descriptions self-documenting and easy to change. The language should allow the designer to specify only selected attributes. This requires automatically supplying default values for all attributes needed for execution of a software prototype. The language should be capable of constructing the software tools in its own prototyping environment.

To support large scale prototypes, system evolution, and parallel execution, a prototyping language should have mechanisms for localizing design decisions in the description and localizing interactions between system components or pieces of knowledge in the knowledge base. These features allow independently designed subsystems of complex expert systems to cooperate without unexpected interference.

To support user validation and system evolution, a prototyping language should support a facility for maintaining the correspondence between requirements and design decisions. Tools will be needed for determining which parts of a description must be removed or modified when a requirements change removes the support for previously made design decisions. The language should provide a harmonious interface to such tools.

Facilities in a prototyping language for recording black-box specifications provides the benefits of a specification language. They support prototype component documentation, verification via proofs and automated testing, and queries for reusable component retrieval. They also form the basis for automated synthesis capabilities, inheritance of common properties and constraints, and consistency checking. For expressiveness, this part of the language may contain non-computable constructs such as quantifiers ranging over unbounded sets. The language should support facilities for describing clear-box characteristics of designs such as interconnections of available components, dependencies between components, design goals such as invariant constraints or bounding functions, and design justifications such as criteria for choosing between alternative designs.

The language should have a distinguished executable subset that is easily recognizable, both by human users and automated tools. Every expression in this distinguished subset should be executable for all possible initial conditions, although some expressions may denote non-terminating computations. The distinguished subset need not contain all of the executable expressions, and expressions outside the distinguished subset may be partially executable, in the sense that execution may fail under some conditions. It should be possible to either augment or transform expressions of the language outside the executable subset to make them executable.

Besides supporting queries for retrieval of reusable software components, the language should have facilities for adapting components to new uses and making small perturbations on their behavior without examining the details of the internal implementation of the components.

To support high productivity, the language should support the construction of efficient implementations by augmenting the prototype description with annotations describing additional constraints or lower level design decisions. This enables the designer to view optimization as a refinement step where additional information is added to the original descriptions, rather than a complete reformulation of the system description. Such an approach saves designer time by avoiding repeated treatment of the same issues in different ways, and by reducing the opportunities for making transcription or translation errors.

Efficiency is more of a concern for the production version of the system than for the prototype, but it cannot be ignored because it must be possible to run test cases and gather data in a reasonable amount of time. This implies that execution mechanisms based on exhaustive enumeration are insufficient to meet the requirements of a prototyping language, although they may be supplied as a default to allow running small test cases in the absence of information about more efficient execution strategies. The language should therefore provide a set of fairly efficient execution mechanisms, tools for locating performance bottlenecks in larger systems, and incremental optimization transformations to improve prototypes that are impractically slow.

Real-time constraints impose a slightly different set of subgoals: execution times must be predictable, although not necessarily very fast. Prototypes of real-time systems may operate in simulated time or linearly scaled real time, but the actual execution times for the production version must be predictable within accurate bounds. The presence of real-time constraints severely restricts the kinds of computations a system may perform, and in the case of expert systems, limits the amount of logical inference that can be performed. The design of expert systems that operate within real-time constraints is a largely unexplored area, and significant research progress is needed in this area to fully realize the goals of a comprehensive prototyping language.

The rapid construction of software prototypes depends on simplifying the view of the system through which the specifiers and designers do their work, and providing automated means for bridging the gap between this simplified view and the detailed programming level description currently needed to make a software system efficiently executable. This automated support should include mechanisms for execution, static analysis of the properties of the proposed system, preparation of test cases, reporting and analyzing results, and diagnosing ill-formed descriptions and departures from desired behavior to allow the specifiers and designers to work entirely within the simplified view, at least during the construction of the initial prototype. While the construction of the tools is not a required until later phases of the project, supporting the construction of such a tool set is a major driving force for the language design. The development of an integrated set of tools requires a consistent and simple semantic model rich enough to express and support all of these functions. Finding suitable models is the key to the project.

3. Modeling Issues

The models underlying the language provide the common ground for the associated set of tools. The semantic model for the language provides the basis for automated analysis, while the computational model provides the basis for execution. One of the main challenges in this project is to find a model that can coherently span the range of applications required. This will require a significant advance in the state of the art.

There is no single common model of expert systems available for rapid prototyping. First order logic is one of the most familiar models for reasoning, but it has been criticized for its weaknesses, such as lack of facilities for handling uncertain information, representing heuristic methods for speeding up conclusions, and non-monotonic reasoning. Many other kinds of logic have been proposed, but the theories of these logics are still being explored and there has been no consensus on whether there is a single logic suitable for constructing all types of expert systems, or which variety of logic is the most promising. There are also approaches to expert systems that are based on models other than logic, such as semantic networks, Bayesian statistics, and production systems. Since it is not clear which approach will yield the best results in the long run, a comprehensive prototyping language must find a unified way of treating most of the issues raised by this diverse set of models.

There is also no single commonly accepted model for representing real-time constraints. Some approaches that have been explored include temporal logic, state machines, mode charts, augmented data flow diagrams, Petri nets, and I/O automata. The model for a comprehensive prototyping language should be chosen to enhance the application of recent results in logic, graph theory, and combinatorics to link the semantic model to an effective execution mechanism. Other unexplored areas include effective models for real-time databases and real-time communications networks. In both of these areas, the problems of providing service within guaranteed worst-case time bounds are largely unexplored.

4. Supporting the Design of Expert Systems

Several special purpose systems for supporting the design of expert systems have been developed, some of which are known as expert system shells. A comprehensive prototyping language should improve on available facilities if possible, and integrate them with facilities suitable for producing other kinds of software. There are two ways of supporting the prototyping of expert systems: (1) adding special purpose features to the prototyping language, and (2) adding predefined reusable software components that can be defined within a general purpose language, such as specialized data types, state machines, and functions. The predefined component approach is preferred to the addition of specialized language features because of the requirement for simplicity. However, such predefined components should have standardized interfaces to improve portability.

Many of the standard building blocks for expert systems can be provided as generic predefined components. These include facts, rules, patterns, frames, contexts, constraints, demons, instance generators, pattern matchers, unification mechanisms, and forward and backward chaining inference engines. Standardization requires careful analysis of these components and specification of their required properties. An open issue is whether current mechanisms for defining generic components are flexible enough to adequately capture the range of behavior required for these kinds of components, and if not, what extensions are required.

Some of the requirements for a prototyping language are determined by the need for prototyping expert systems. Examples of such requirements are

(1) a means for conveniently defining external representations and input facilities for the knowledge in the knowledge base,

(2) support for the first class treatment of higher order objects such as types, functions, tasks, and generators, and

(3) support for control mechanisms such as state-triggered demons, backtracking, run-time control over task priorities, and scheduling of temporal events.

It is important to meet these requirements in a prototyping language for expert systems.

5. Knowledge Base Issues

Knowledge base management is an important part of the design of expert systems. Rapid prototyping of knowledge based systems brings special requirements for the design of a prototyping language as well as its environment. Expert system technology is useful in implementing parts of the supporting environment for a prototyping language. For example, such an environment needs knowledge base support for the following functions:

(1) Managing reusable components. The environment should contain a large software base with reusable components. This software base should be coupled with a set of rules for tailoring and combining available components to fulfill queries that do not exactly match any of the components explicitly stored in the software base.

(2) High level debugging. Errors and failures during prototype execution should be mapped from the programming language level to level of the prototyping language, to allow the designer to work entirely in terms of the semantic model associated with the prototyping language.

(3) Optimization. The transformations for optimizing a prototype version of a system to produce a production version should be performed with minimum interaction with the designer. This implies keeping track of the decisions made by the designer in optimizing previous versions of the system, determining which of those decisions are still valid for later versions, and automatically applying the ones that are found to be still valid.

(4) Explanations. Justifications for decisions made automatically should be available to provide feedback to the designer in cases where automated design completion procedures fail. This requires an expert system with a substantial knowledge base.

These needs indicate that the prototyping system associated with a comprehensive prototyping language will need expert system technology for realizing some of its major subsystems.

6. Conclusions

DARPA/ISTO has made an important decision to develop designs for a rapid prototyping language for software systems, which is intended to apply to a variety of large software systems, including knowledge-based systems, parallel systems, distributed systems, and real-time systems. The Common Prototyping Language project has an ambitious set of goals that raises many interesting research problems, many of which involve expert systems. Solutions to these problems are essential for achieving significant improvements in the quality and productivity of the software development process.

The proposed connection between the Common Prototyping Language and Ada raises several issues that must be considered. Goals for the Common Prototyping Language include computer-aided transformations of prototypes into Ada implementations of the production version of the software, and eventually implementing the tools in the prototyping system in Ada to provide portability. This poses a problem

because ordinary compiler technology is insufficient for execution of the prototyping language. The need for flexibility and run-time handling of newly created types and procedures to support expert systems also provides challenges for efficient implementation techniques in terms of Ada. Conventional translation techniques must be coupled with facilities for scheduling to meet hard real-time constraints with transformations to allow the execution of incompletely specified processes, and access to an interpreter or an incremental compiler at run-time.

Ada provides a completely static type system, treats types and functions as second class objects, and requires task priorities to be known at compilation time. It is clear that the flexibility required for supporting expert systems development can be provided by adding a run-time interpreter on top of the Ada language. The difficult problem will be to provide these features efficiently, and without introducing excessive run-time overheads for portions of the prototype that do not require flexibility beyond that provided directly by Ada.

Ada provides relatively weak guarantees about the scheduling of tasks, and limits programmer control over scheduling to statically specified priorities. Since this is somewhat removed from the level of support needed for implementing hard real-time systems, the execution support system for the prototyping language will have to provide higher level facilities for scheduling real-time operations. Such facilities can be classified as on-line (done at run-time) and off-line (done prior to execution). There is no universally accepted approach to real-time scheduling. Optimal scheduling algorithms are very time consuming, and generally cannot be carried out on-line, while off-line approaches are inflexible and do not handle overload situations very well. There are many different scheduling algorithms, and choosing the best one for a given application is a difficult problem.

Transformations are needed to execute incompletely specified components. Such transformations should supply reasonable default values for attributes necessary for execution if the designer does not explicitly specify them. Such attributes can be explicitly specified to produce a more accurate model of the system or to improve its performance. One example of such attributes is the assignment of tasks to physical processors. Sometimes the assignment of particular critical tasks to particular processors is necessary to meet tight timing constraints by avoiding the overhead of some interprocessor communication. However, the designer usually does not care about the placement of all tasks, and would like the system to assign reasonable default locations to all of the tasks that do not have explicit processor assignments.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Office of Naval Research Office of the Chief of Naval Research Attn. CDR Michael Gehl, Code 1224 800 N. Quincy Street Arlington, Virginia 22217-5000	1
4.	Space and Naval Warfare Systems Command Attn. Dr. Knudsen, Code PD 50 Washington, D.C. 20363-5100	1
5.	Ada Joint Program Office OUSDRE(R&AT) Pentagon Washington, D.C. 20301	1
6.	Naval Sea Systems Command Attn. CAPT Joel Crandall National Center #2, Suite 7N06 Washington, D.C. 22202	1
7.	Office of the Secretary of Defense Attn. CDR Barber STARS Program Office Washington, D.C. 20301	1
8.	Office of the Secretary of Defense Attn. Mr. Joel Trimble STARS Program Office Washington, D.C. 20301	1
9.	Commanding Officer Naval Research Laboratory Code 5150 Attn. Dr. Elizabeth Wald Washington, D.C. 20375-5000	1

10. Navy Ocean System Center — 1
Attn. Linwood Sutton, Code 423
San Diego, California 92152-500

11. National Science Foundation — 1
Attn. Dr. William Wulf
Washington, D.C. 20550

12. National Science Foundation — 1
Division of Computer and Computation Research
Attn. Dr. Peter Freeman
Washington, D.C. 20550

13. National Science Foundation — 1
Director, PYI Program
Attn. Dr. C. Tan
Washington, D.C. 20550

14. Office of Naval Research — 1
Computer Science Division, Code 1133
Attn. Dr. Van Tilborg
800 N. Quincy Street
Arlington, Virginia 22217-5000

15. Office of Naval Research — 1
Applied Mathematics and Computer Science, Code 1211
Attn: Dr. James Smith
800 N. Quincy Street
Arlington, Virginia 22217-5000

16. New Jersey Institute of Technology — 1
Computer Science Department
Attn. Dr. Peter Ng
Newark, New Jersey 07102

17. Southern Methodist University — 1
Computer Science Department
Attn. Dr. Murat Tanik
Dallas, Texas 75275

18. Editor-in-Chief, IEEE Software — 1
Attn. Dr. Ted Lewis
Oregon State University
Computer Science Department
Corvallis, Oregon 97331

19. University of Texas at Austin — 1
Computer Science Department
Attn. Dr. Al Mok
Austin, Texas 78712

20. University of Maryland — 1
College of Business Management
Tydings Hall, Room 0137
Attn. Dr. Alan Hevner
College Park, Maryland 20742

21. University of California at Berkeley — 1
Department of Electrical Engineering and Computer Science
Computer Science Division
Attn. Dr. C.V. Ramamoorthy
Berkeley, California 94720

22. University of California at Los Angeles — 1
School of Engineering and Applied Science
Computer Science Department
Attn. Dr. Daniel Berry
Los Angeles, California 90024

23. University of Maryland — 1
Computer Science Department
Attn. Dr. Y. H. Chu
College Park, Maryland 20742

24. University of Maryland — 1
Computer Science Department
Attn. Dr. N. Roussapoulos
College Park, Maryland 20742

25. Kestrel Institute — 1
Attn. Dr. C. Green
1801 Page Mill Road
Palo Alto, California 94304

26. Massachusetts Institute of Technology — 1
Department of Electrical Engineering and Computer Science
545 Tech Square
Attn. Dr. B. Liskov
Cambridge, Massachusetts 02139

27. Massachusetts Institute of Technology — 1
Department of Electrical Engineering and Computer Science
545 Tech Square
Attn. Dr. J. Guttag
Cambridge, Massachusetts 02139

28. University of Minnesota — 1
Computer Science Department
136 Lind Hall
207 Church Street SE
Attn. Dr. J. Ben Rosen
Minneapolis, Minnesota 55455

29. International Software Systems Inc. — 1
12710 Research Boulevard, Suite 301
Attn. Dr. R. T. Yeh
Austin, Texas 78759

30. Software Group, MCC — 1
9430 Research Boulevard
Attn. Dr. L. Belady
Austin, Texas 78759

31. Carnegie Mellon University — 1
Software Engineering Institute
Department of Computer Science
Attn. Dr. Lui Sha
Pittsburgh, Pennsylvania 15260

32. IBM T. J. Watson Research Center — 1
Attn. Dr. A. Stoyenko
P.O. Box 704
Yorktown Heights, New York 10598

33. The Ohio State University — 1
Department of Computer and Information Science
Attn. Dr. Ming Liu
2036 Neil Ave Mall
Columbus, Ohio 43210-1277

34. University of Illinois — 1
Department of Computer Science
Attn. Dr. Jane W. S. Liu
Urbana Champaign, Illinois 61801

35. University of Massachusetts — 1
Department of Computer and Information Science
Attn. Dr. John A. Stankovic
Amherst, Massachusetts 01003

36. University of Pittsburgh — 1
Department of Computer Science
Attn. Dr. Alfs Berztiss
Pittsburgh, Pennsylvania 15260

37. Defense Advanced Research Projects Agency (DARPA) — 1
Integrated Strategic Technology Office (ISTO)
Attn. Dr. Jacob Schwartz
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

38. Defense Advanced Research Projects Agency (DARPA) 1
Integrated Strategic Technology Office (ISTO)
Attn. Dr. Squires
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

39. Defense Advanced Research Projects Agency (DARPA) 1
Integrated Strategic Technology Office (ISTO)
Attn. MAJ Mark Pullen, USAF
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

40. Defense Advanced Research Projects Agency (DARPA) 1
Director, Naval Technology Office
1400 Wilson Boulevard
Arlington, Virginia 2209-2308

41. Defense Advanced Research Projects Agency (DARPA) 1
Director, Strategic Technology Office
1400 Wilson Boulevard
Arlington, Virginia 2209-2308

42. Defense Advanced Research Projects Agency (DARPA) 1
Director, Prototype Projects Office
1400 Wilson Boulevard
Arlington, Virginia 2209-2308

43. Defense Advanced Research Projects Agency (DARPA) 1
Director, Tactical Technology Office
1400 Wilson Boulevard
Arlington, Virginia 2209-2308

44. MCC AI Laboratory 1
Attn. Dr. Michael Gray
3500 West Balcones Center Drive
Austin, Texas 78759

45. COL C. Cox, USAF 1
JCS (J-8)
Nuclear Force Analysis Division
Pentagon
Washington, D.C. 20318-8000

46. LTCOL Kirk Lewis, USA 1
JCS (J-8)
Nuclear Force Analysis Division
Pentagon
Washington, D.C. 20318-8000

47. University of California at San Diego — 1
Department of Computer Science
Attn. Dr. William Howden
La Jolla, California 92093

48. University of California at Irvine — 1
Department of Computer and Information Science
Attn. Dr. Nancy Levenson
Irvine, California 92717

49. University of California at Irvine — 1
Department of Computer and Information Science
Attn. Dr. L. Osterweil
Irvine, California 92717

50. University of Colorado at Boulder — 1
Department of Computer Science
Attn. Dr. Lloyd Fosdick
Boulder, Colorado 80309-0430

51. Santa Clara University — 1
Department of Electrical Engineering and Computer Science
Attn. Dr. M. Ketabchi
Santa Clara, California 95053

52. Oregon Graduate Center — 1
Portland (Beaverton)
Attn. Dr. R. Kieburtz
Portland, Oregon 97005

53. Dr. Wolfgang Halang — 1
Bayer AG
Ingenieurbereich Progessleittechnik
D-4047
Dormagen, West Germany

54. Dr. Bernd Kraemer — 1
GMD Postfach 1240
Schloss Birlinghaven
D-5205
Sankt Augustin 1, West Germany

55. Dr. Aimram Yuhudai — 1
Tel Aviv University
School of Mathematical Sciences
Department of Computer Science
Tel Aviv, Israel 69978

56. Dr. Robert M. Balzer 1
USC-Information Sciences Institute
4676 Admiralty Way
Suite 1001
Marina del Ray, California 90292-6695

57. U.S. Air Force Systems Command 1
Rome Air Development Center
RADC/COE
Attn. Mr. Samuel A. DiNitto, Jr.
Griffis Air Force Base, New York 13441-5700

58. U.S. Air Force Systems Command 1
Rome Air Development Center
RADC/COE
Attn. Mr. William E. Rzepka
Griffis Air Force Base, New York 13441-5700

59 LuQi 50
Code 52Lq
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100

60. Research Administration 1
Code: 012
Naval Postgraduate School
Monterey, CA. 93943